Contents

Egress TLS Origination

6 minute read ✓ page test

The Accessing External Services task demonstrates how external, i.e., outside of the service mesh, HTTP and HTTPS services can be accessed from applications inside the mesh. As described in that task, a `ServiceEntry` is used to configure Istio to access external services in a controlled way. This example shows how to configure Istio to perform TLS origination for traffic to an external service. Istio will open HTTPS connections to the external service while the original traffic is HTTP.

Use case

Consider a legacy application that performs HTTP calls to external sites. Suppose the organization that operates the application receives a new requirement which states that all the external traffic must be encrypted. With Istio, this requirement can be achieved just by configuration, without changing any code in the application. The application can send unencrypted HTTP requests and Istio will then encrypt them for the application.

Another benefit of sending unencrypted HTTP requests from the source, and letting Istio perform the TLS upgrade, is that Istio can produce better telemetry and provide more routing control for requests that are not encrypted.

Before you begin

- Setup Istio by following the instructions in the Installation guide.
- Start the sleep sample which will be used as a test source for external calls.

  If you have enabled automatic sidecar injection, deploy the sleep application:

  ```
  $ kubectl apply -f @samples/sleep/sleep.yaml@
  ```

  Otherwise, you have to manually inject the sidecar before deploying the sleep application:

```
$ kubectl apply -f <(istioctl kube-inject -f @samples/sleep/sleep.yaml@)
```

Note that any pod that you can `exec` and `curl` from will do for the procedures below.

- Create a shell variable to hold the name of the source pod for sending requests to external services. If you used the `sleep` sample, run:

```
$ export SOURCE_POD=$(kubectl get pod -l app=sleep -o jsonpath={.items..metadata.name})
```

Configuring access to an external service

First start by configuring access to an external service, `edition.cnn.com`, using the same technique shown in the Accessing External Services task. This time, however, use a single `ServiceEntry` to enable both HTTP and HTTPS access to the service.

1. Create a `ServiceEntry` to enable access to `edition.cnn.com`:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: edition-cnn-com
spec:
  hosts:
  - edition.cnn.com
  ports:
  - number: 80
    name: http-port
    protocol: HTTP
  - number: 443
    name: https-port
    protocol: HTTPS
  resolution: DNS
EOF
```

2. Make a request to the external HTTP service:

```
$ kubectl exec "${SOURCE_POD}" -c sleep -- curl -sSL -o /de
v/null -D - http://edition.cnn.com/politics
HTTP/1.1 301 Moved Permanently
...
location: https://edition.cnn.com/politics
...

HTTP/2 200
...
```

The output should be similar to the above (some details replaced by ellipsis).

Notice the `-L` flag of *curl* which instructs *curl* to follow redirects. In this case, the server returned a redirect response (`301 Moved Permanently`) for the HTTP request to `http://edition.cnn.com/politics`. The redirect response instructs the client to send an additional request, this time using HTTPS, to `https://edition.cnn.com/politics`. For the second request, the server returned the requested content and a *200 OK* status code.

Although the *curl* command handled the redirection transparently, there are two issues here. The first issue is the redundant request, which doubles the latency of fetching the content of `http://edition.cnn.com/politics`. The second issue is that the path of the URL, *politics* in this case, is sent in clear text. If there is an attacker who sniffs the communication between your application and `edition.cnn.com`, the attacker would know which specific topics of `edition.cnn.com` the application fetched. For privacy reasons, you might want to prevent such disclosure.

Both of these issues can be resolved by configuring Istio to perform TLS origination.

TLS origination for egress traffic

1. Redefine your `ServiceEntry` from the previous section to redirect HTTP requests to port 443 and add a `DestinationRule` to perform TLS origination:

    ```
    $ kubectl apply -f - <<EOF
    apiVersion: networking.istio.io/v1alpha3
    kind: ServiceEntry
    metadata:
      name: edition-cnn-com
    spec:
      hosts:
      - edition.cnn.com
      ports:
      - number: 80
        name: http-port
        protocol: HTTP
        targetPort: 443
      - number: 443
        name: https-port
    ```

```
    protocol: HTTPS
  resolution: DNS
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: edition-cnn-com
spec:
  host: edition.cnn.com
  trafficPolicy:
    portLevelSettings:
    - port:
        number: 80
      tls:
        mode: SIMPLE # initiates HTTPS when accessing edition.cnn.com
EOF
```

The above `DestinationRule` will perform TLS origination for HTTP requests on port 80 and the `ServiceEntry` will then redirect the requests on port 80 to target port 443.

2. Send an HTTP request to `http://edition.cnn.com/politics`, as in the previous section:

   ```
   $ kubectl exec "${SOURCE_POD}" -c sleep -- curl -sSL -o /de
   v/null -D - http://edition.cnn.com/politics
   HTTP/1.1 200 OK
   ...
   ```

   This time you receive *200 OK* as the first and the only response. Istio performed TLS origination for *curl* so the original HTTP request was forwarded to `edition.cnn.com` as HTTPS. The server returned the content directly, without the need for redirection. You eliminated the double round trip between the client and the server, and the request left the mesh encrypted, without disclosing the fact that your application fetched the *politics* section of `edition.cnn.com`.

Note that you used the same command as in the previous section. For applications that access external services programmatically, the code does not need to be changed. You get the benefits of TLS origination by configuring Istio, without changing a line of code.

3. Note that the applications that used HTTPS to access the external service continue to work as before:

```
$ kubectl exec "${SOURCE_POD}" -c sleep -- curl -sSL -o /dev/null -D - https://edition.cnn.com/politics
HTTP/2 200
...
```

Additional security considerations

Because the traffic between the application pod and the sidecar proxy on the local host is still unencrypted, an attacker that is able to penetrate the node of your application would still be able to see the unencrypted communication on the local network of the node. In some environments a strict security requirement might state that all the traffic must be encrypted, even on the local network of the nodes. With such a strict requirement, applications should use HTTPS (TLS) only. The TLS origination described in this example would not be sufficient.

Also note that even with HTTPS originated by the application, an attacker could know that requests to `edition.cnn.com` are being sent by inspecting Server Name Indication (SNI). The *SNI* field is sent unencrypted during the TLS handshake. Using HTTPS prevents the attackers from knowing specific topics and articles but does not prevent an attackers from learning that `edition.cnn.com` is accessed.

Cleanup

1. Remove the Istio configuration items you created:

```
$ kubectl delete serviceentry edition-cnn-com
$ kubectl delete destinationrule edition-cnn-com
```

2. Shutdown the `sleep` service:

```
$ kubectl delete -f @samples/sleep/sleep.yaml@
```